The BRAIN-CA™ Estimator and The Cincinnati Algorithm: Simplification Tools for Artificial Intelligence

Jerry Felix, Steve Brunker, Carol Hibbard
Brain-CA Technologies, Inc.

Learning is a process of creating a model of information as it is observed. The ideal storage format of such a model would represent learned information so that the model could be rapidly applied or modified, as additional observations are made. The BRAIN-CA™ Estimator is an elemental tool which enables this capability: the storage of a simple model in a manner that is accessible for rapid modification (learning) and rapid application (inference). A process called The Cincinnati Algorithm provides a simple method to update the Estimator model as additional observations are made.

Biological systems and AI systems learn by creating a model of information that is observed. In the case of biological systems, the brain is bombarded every millisecond with sensory information. Each bit of sensory information must be processed rapidly as it arrives, because additional information is immediately following. An ideal learning system is able to rapidly adjust its model given newly observed information. Brain-CA Technologies designed a simple format for a data model that easily adjusts to reflect new information. In addition, the model's format allows for rapid application of what has been learned.

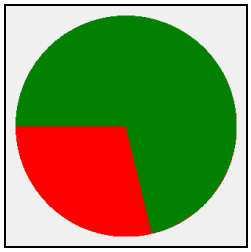
To build an elemental component, we focused on creating a model format and modification process for the simplest challenge we could define - modeling a binary data stream. Our focus was to learn and store the most basic characteristic about such a stream: the ratio of one binary value to another.

The ability to characterize a binary data stream is an inherent capability of the human brain that is so fundamental that it is easily overlooked. As a demonstration of this, imagine you are watching a light that blinks a color, seemingly at random - either red or green. Suppose you are provided no advanced instruction, and you watch that light for a while. In this demonstration, it blinks one color about twice as often as the other. Even without advance notice that you'd be asked, you would be able to reply correctly when asked to identify the predominant color.

Somehow, your brain observes and maintains an estimate of the ratio of one binary event compared to another. And your brain does this automatically with *any* binary situation - whether it's observing red or green lights, experiencing pain or pleasure, or feeling wet or dry. It's unlikely that your brain is performing math (counting and dividing) to achieve this feat. It is also unlikely that each color (or binary option) is stored for later analysis.

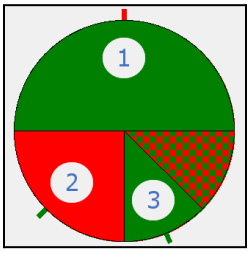
We hypothesize that a *model* is automatically created by the brain to rapidly process the data stream in real time. And your brain adjusts its model swiftly as additional information arrives. When asked to identify the predominant color, you are able to apply the model quickly without further analysis. This is the biological process that we set out to emulate, when designing the **BRAIN-CA™ Estimator.**

The BRAIN-CA™ Estimator as a Pie Chart



The biological brain is able to accurately estimate with some granularity. A pie chart is a good visual model of this phenomenon. The brain can discern the difference between "almost always red", "usually red", "about 50/50", "usually green", and "almost always green". Accuracy of the estimate is reduced at higher granularity; you aren't likely to be able to discern the difference between 33% and 34%, but you can probably tell the difference between 33% and 50%.

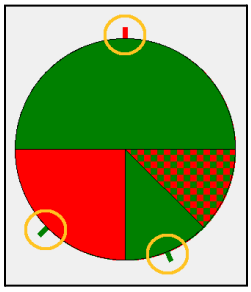
We set out to design a tool to emulate this feature of the brain. We envisioned a lightweight model. A model's representation should generally require fewer data bits than the data that is being modeled. And the model should be easy to adjust, as new data arrives.



Our solution is to construct a virtual pie chart, piece by piece. The figure to the left is a visual depiction of an Estimator that has three pie pieces, and a fourth area that is not yet defined, represented by a red/green checkered slice of pie.

The model on the left is using three storage bits; the first bit stores a representation of the binary value that was observed most, and it represents half the pie. The second bit is a quarter of the pie. As additional observations are made, the pie pieces adjust according to a simple set of rules that we have named **The Cincinnati Algorithm**. The rules may adjust the colors of the existing pieces of pie or place additional pieces. Each additional piece of pie will consume half of the undefined (checkered) area.

We noted that in a binary situation, at least one of the two choices must be a minimum of half of the pie. And if you disregard the most common half of the observations, at least one of the choices must be half of the remainder, and so on. It is the Estimator's job to model the data as it arrives. If, at any time, you want to identify the overall predominant color, simply look at bit number one.



Another feature of the Estimator is that each storage bit is paired with a random bit, depicted on the pie chart as tiny lines, and circled in the image on the right. The random bits may vary any time an observation is made.

The patented pairing of a storage bit and a random bit is the key to the Estimator's ability to model data streams.[1] By pairing each storage bit with a random bit, the Estimator is able to rapidly adjust its model in a probabilistic manner, as new observations are made. Random bits can be generated using electronic noise.[2]

---

[1] US Patent 11,847,386 B1. https://patents.google.com/patent/US11847386B1/ Other patents pending.

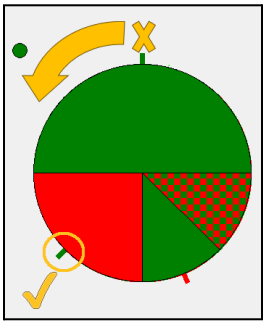
[2] Stipcevic, "Non-deterministic Random Bit Generator Based on Electronics Noise", 2008, arXiv:physics/0309010v2 [physics.comp-ph]. https://arxiv.org/abs/physics/0309010.

The Cincinnati Algorithm

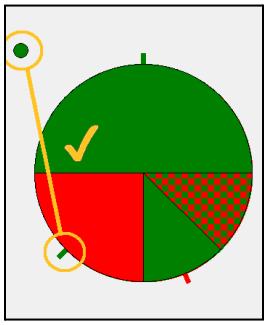
As additional observations arrive, the Estimator is adjusted or extended through a series of simple steps, using basic logic functions - bit comparisons and inversions;[3] no arithmetic operations are necessary. A ratio - which is typically considered to be a *quotient* - is converged upon as observations accumulate, without executing a *divide* operation.

The Cincinnati Algorithm

1. Initialize the Estimator as a null bit string.
2. Pair each bit of the Estimator with a random bit.
3. Observe a binary value (0 or 1, Red or Green, etc.).
4. Starting at the high order bit, find an Estimator bit that doesn't match its random bit.
5. If all match, then extend the Estimator with the observed bit.
6. Otherwise, if the unmatched random bit matches the observed bit, invert the low order Estimator bits until one is inverted to be the same as the observed bit.
7. Return to step 2.



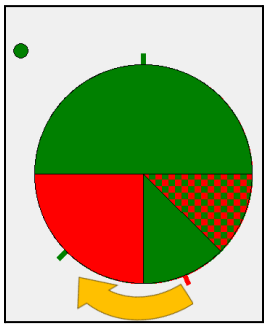
In the example that follows, an Estimator has previously observed at least three (and possibly many more) flashes of the light. Its state at the time of the next light flash is shown to the left. The light being observed is the small green circle in the upper left.

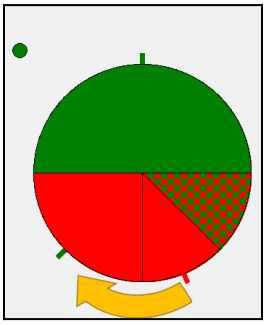


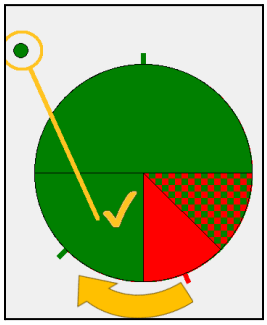
In step 3 of The Cincinnati Algorithm, we observe that light. And step 4, we find the largest pie piece with a mis-matched random bit. In this case, it's the second largest piece, the red one.

Step 5, since we found a mismatch, we proceed to step 6. We compare the mismatched random bit to the observed bit, as shown to the right. In this case, they are both green - they match, so we begin the inversion process.

We "invert" the pie pieces, beginning at the smallest piece, and continue to invert until one is inverted to be the same as the observation, as shown in the images below, left-to-right. The first image shows the initial state. The second image inverts the smallest pie piece. The third inverts the second pie piece, which becomes the same color as the light, ending the inversion process.



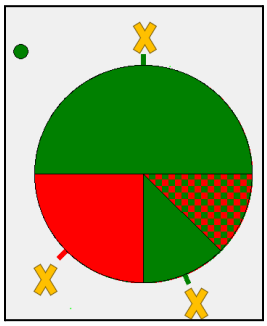




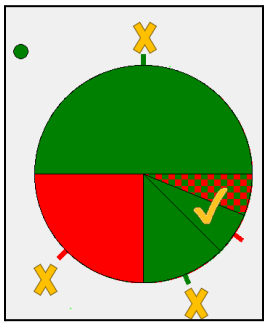
[3] In the context of the pie chart, a bit inversion is changing a red pie piece to green and a green pie piece to red.

The final disposition, after The Cincinnati Algorithm Step 6 adjustment, has two larger green pieces, and one red piece. Effectively, the Estimator transitioned from an initial state of three pie pieces, which, largest to smallest, were Green - Red - Green, to a state of Green - Green - Red. By observing a green light, the estimated probability of green increased slightly (that is, there is more green in the final pie chart than the initial one), and red decreased slightly.

Walking through another example, let's examine what happens when no mismatched random bits are found in step 5. We'll start in the same initial state; from largest to smallest piece: Green - Red - Green.



As shown to the left, at step 4, no mismatch is found between the random bits and the Estimator storage bits. So we proceed in step 5 to add a pie piece, shown to the right, which is the same color as the observed light.



In this example, the Estimator observed a green light, and transitioned from Green - Red - Green, to Green - Red - Green - Green. Again, our estimate of the proportion of green increased slightly after observing a green light.

Note that as the number of observations increases, the length of the Estimator increases logarithmically; for each bit of Estimator length, the probability of having no mismatches gets cut in half.

There is also a case, not shown, where there is no adjustment to the Estimator when making an observation. This happens when there's a mismatched random bit in step 4, but the random bit is not the same as the observation. This can be thought of as the case where we have an estimate of the ratio, and the observation seems to be consistent with this estimate.

To understand this behavior, consider the extreme: if the Estimator is entirely one color - that is, it's made up of, say, all green pie pieces (all green Estimator bits) - and there exists a mismatch between at least one of the green Estimator bits and its paired random bit, then that random bit must be red. And if that random bit doesn't match the observation, then the observation must be green, which is the overwhelmingly predominant color of our pie chart. Since things happened as expected (we expected green, overwhelmingly, and the observation indeed was green), we wouldn't make an adjustment.

On the contrary, if all the pie pieces are green and we observe a red, then things didn't happen as expected, and we're sure to either adjust the Estimator (inverting the smallest piece to be red) or extend to another pie piece (which would be red). Either way, we slightly increase our estimate of red.

Comparison with counting and division method

There are several other common techniques that the Estimator compares favorably against. One traditional method is to simply count the number of red occurrences and the number of green occurrences and divide by the total number of occurrences. This would provide a more accurate ratio of the observations.

Keep in mind, however, that the observations are actually a random sample of an underlying ratio, and it is the underlying ratio that we're trying to estimate. For instance, if the data stream is randomly generating red and green flashes, at a ratio of 2 reds to 5 greens, then the counts might reflect values that

are slightly different (or even significantly different) than the underlying 2:5 proportion. 2 out of 7 equates to approximately 28.5714%. Out of, say, 219 observations, you might expect to see 62.6 reds, but you may only see 57 just due to random chance. A calculated ratio using counters would provide an answer of 57 divided by 219, or roughly 26.0274% (as compared to the underlying ratio of 28.5714%). The precision of the ratio of the counting method is misleading and unnecessary.

Consider the use case of batting averages in baseball. Traditionally, they are provided as a three digit number representing a ratio of hits to at-bats. If a batter has two hits in seven at-bats, they would be reported to have a batting average of .286.

Assume for a moment that there is an underlying fundamental, unchanging skill that a hitter has, and our goal is to estimate that capability. The seven at-bats are samples of that skill. It would be misleading and unnecessary to store that estimated skill rating as 0.285714285714… There have only been seven samples! It seems better to provide a rating of "about 0.3". If later, the player achieves 14 hits in 98 at-bats (i.e. the same 2/7ths ratio), we might provide a rating of "about 0.29". And subsequently, if they are 142 for 994, we might use "about 0.286". We can effectively encode both the estimated ratio *and* the approximate number of samples into one value. The Estimator does exactly that, in binary representation.

The initial state of an Estimator is null (zero bits), meaning that we have no information about the data stream. In a pie chart visualization, the entire pie might be checkered red and green, depicting a 50/50 estimate. After the first observation, half the pie will be that which was observed. This can be thought of as 50% one color, with the other 50% checkered (or split 50/50 between the choices). So after one observation, our estimate can be thought of as 75/25, with the 75 being the color that was observed.[4]

With one bit, the Estimator is able to store values of one fourth and three fourths. Two bits allow the estimate to vary in eighths (that is, 1, 3, 5, or 7 eighths). And so on. In an alternate implementation (where the undefined section of pie is disregarded), a one-bit Estimator can store values of 0 or 1; two bits allow the estimate to vary in thirds (that is, 0, 1, 2 or 3 thirds); three bits allow the estimate to vary in sevenths (0 through 7 sevenths). The mechanics of The Cincinnati Algorithm are the same in both implementations; it's just the interpretation of the results that varies. The Estimator models the observations by adjusting probabilistically, storing limited information, and converging over time to the exact ratio.

In the batting average example shown above, if we are only storing one value to represent both the estimated ratio and an encoding of the order of magnitude of the number of samples, then the issue arises of when to increase the number of digits, particularly if we want the number of digits to represent an order of magnitude of the number of at-bats (10, 100, 1000, etc.). Two digits might represent approximately 100 at bats, as an order of magnitude, with a probability distribution around 100. But if we are not storing the actual count of at-bats, then when do we advance to three digits? The same issue arises with the binary Estimator representation. Our solution is to increase the number of bits probabilistically, in steps 4 and 5 of The Cincinnati Algorithm. The Estimator has a lower probability of extending as it gets longer.[5]

---

[4] In some implementations, it may be useful to disregard the undefined section of the pie, or consider it to be divided in the same proportions as the rest of the pie.

[5] This is similar to Robert Morris' Approximate Counting algorithm: Morris, R. *Counting large numbers of events in small registers*. Communications of the ACM 21, 10 (1978), 840–842

While counting and division are trivial operations for today's processors, the Estimator is a much simpler device. This is relevant in cases where logic minimization is critical. Specifically, in AI applications where massive parallelism is desired, reduction of complexity is critical when the individual component is replicated millions or billions of times in silicon (such as Cellular Automata systems).

Comparison with methods currently used in AI

A common technique used in state of the art AI systems is to use back propagation in an effort to minimize a cost function by adjusting a neural network's weights and biases. The level of adjustment is typically determined by gradient descent of the cost function with respect to the parameter. The gradient is calculated via the derivative of a cost function with respect to the change in the parameter variable. In other words, a lot of calculus and floating point operations.

While current AI methods are mathematically sound, and their results speak for themselves, they are complex and expensive. The BRAIN-CA™ Estimator and The Cincinnati Algorithm attempt to simplify parameter adjustments by eliminating floating point operations, and instead replacing them with simple bit manipulations: compare and invert.

Darwinian evolution concepts suggest that simple traits that are useful are more likely to emerge first in a species, rather than complex traits. The simplicity of Estimator technology leads us to believe that it is likely more similar to mechanics of the biological brain than artificial neural network technology involving floating point operations. Bit manipulations (invert and compare) are far simpler than derivatives, and even simpler than counting and dividing.

Model Usage

The BRAIN-CA™ Estimator is designed to swiftly adjust the model with new observations. A simple prediction is readily available, since the most frequently seen result is stored in the high order bit. A more sophisticated prediction is also available. Transformative AI creates predictions proportional to the weighting provided by the model. For example, a simple transformative model generating text might arrive at a position to predict the next letter after " th", and it might randomly choose a letter based on the model's knowledge that the next letter is most likely to be an "e" at 40% probability, or an "a" at 20% probability. Transformative AI chooses a token based on the probability, and doesn't always choose the highest probability option.

The Estimator facilitates that sort of prediction as well. Simply choose the most likely value half of the time (and there's a random bit that is conveniently available, paired with that first Estimator bit). If the most likely value is not chosen, then choose the second most likely option half of the remaining time (and there's a random bit paired with that second Estimator bit), and so on.

The Estimator enables the creation of a simple model of a single data stream. While this may appear to be too trivial to provide value, an analogy is a simple universal logic gate, such as NAND. Universal gates perform simple tasks, which have little value as single units. But millions or billions of them can be cleverly arranged to provide powerful logic circuits in today's supercomputers and large AI systems. Clearly, simple fundamental components can have tremendous value.

Brain-CA Technologies, Inc., has patented a way to arrange Estimators in a way that expands their value. Multiple Estimators can be utilized to track correlations between multiple data streams, performing pattern recognition and other AI functions utilizing bit manipulation.[6] This is a simpler way of performing AI functions of learning and inference versus floating point computations and advanced calculus, commonly used in today's neural networks.

To see how this might be possible, consider looking for relationship patterns in two binary streams. The two might be highly correlated, where one data stream value can predict the value of the other. Or they may have little or no correlation. By arranging two Estimators, we can find the relationship. There are only four possible combinations of two binary variables. For example, (Positive, Positive), (Negative, Negative), (Positive, Negative), and (Negative, Positive).

One Estimator may track the second stream's value when the first is Positive, and the other might track the second stream's value when the first is Negative. The two ratios generated by these Estimators can be compared, and if they are substantially different (after a sufficient number of observations), then it can be determined that the first value can predict the second, with some level of accuracy; the second is *dependent* on the first. And if the second generally occurs later in time (or later in *text*, for text based examples such as Large Language Models and other transformative AI), then we have a learning system with an integrated inference engine.

Extended Estimator Functionality

The BRAIN-CA™ Estimator performs well for its specific task, modeling a binary data stream, specifically, determining the ratio of one binary value to another. We then considered additional use cases. For example, whether the Estimator can provide value when the light has a recognizable pattern such as, it repeatedly blinks Red - Red - Green - Red - Red - Green … The biological brain would quickly recognize that pattern.

We found that by using paired Estimators as mentioned in the previous section, and looking for patterns between a data stream and an earlier version of that same data stream, repeating patterns can be recognized. In addition, an oscillating data stream can be utilized to achieve similar results.

We also considered the use case where the underlying ratio changes. Recall in our baseball example, we cited an "underlying fundamental, unchanging skill". We assumed that the ratio doesn't change over time. However, ideally our model will adapt quickly to changes in underlying parameters.

By limiting the number of pie pieces, we are able to introduce a recency bias to allow our model to adapt more quickly than traditional methods. For example, we compared the performance of counters versus Estimators, by training each system with 5000 observations, and limiting the Estimator to six pie pieces (bits). Both the counter method and the Estimator were able to converge on our underlying ratio of 2:5. Then we modified the underlying ratio to be 5:2. On average it will take about 45 subsequent observations for the Estimator to reflect that the predominant color had changed, while the counter will predictably need many more observations to neutralize the original 5000 observations. In effect, by capping the number of pie pieces we created a bias that emphasizes more recent observations.

---

[6] Felix, Brunker, Hibbard: "Casting off the Old Guard: Achieving Superior A.I. Performance through Simplification" 2024. https://brain-ca.com/casting-off-the-old-guard/

Conclusion

Brain-CA Technologies has invented and patented a device called the **BRAIN-CA™ Estimator** and a method called **The Cincinnati Algorithm** to model a single data stream, without the need for mathematical computations. We have identified ways to arrange collections of these devices to address specific AI challenges, such as the time and energy required for model training, costs, and portability.

Our technology emulates an often overlooked capability of the biological brain - the ability to estimate a ratio of a binary event converging over time. We believe that this is an elemental component to future AI systems that will increase energy efficiency as this technology combines learning and inference into a single unified architecture.

About Brain-CA Technologies, Inc.

Founded by three former long-time Hewlett Packard (Enterprise) employees, Brain-CA Technologies incorporated in November 2023, and quickly secured patents on key technologies related to Artificial Intelligence.

Early in the founders' HP careers, HP was a pioneer in a technology called RISC, Reduced Instruction Set Computing. RISC technology is prevalent today, but back then, CISC (Complex Instruction Set Computing) was the norm. The trend from 1950-1985 was to *increase* the capability (and complexity) of the CPU.

Through extensive measurement, HP arrived at a design for simplified CPU chips that could out-perform the complex ones, and worked with Intel to transition the world to RISC computing. Brain-CA Technologies' efforts follow that line of thinking - keep simplifying. The simplicity of our design offers the opportunity for breakthroughs in performance through massive parallelism while decreasing energy consumption.

Authors

Jerry Felix: https://www.linkedin.com/in/jerfelix/
Steve Brunker: https://www.linkedin.com/in/stevebrunker/
Carol Hibbard: https://www.linkedin.com/in/carolhibbard/

Brain-CA Technologies

For additional information:
https://brain-ca.com/
info@brain-ca.com
513.360.8603